A Procedural Control Language for a Digital Signal Processor

Paul Berg
Royal Conservatory
Juliana van Stolberglaan 1
2595 CA The Hague, Netherlands

A procedural control language called Pile4 has been developed and implemented at the Royal Conservatory in The Hague, Netherlands, for a DMX-1000 Signal Processing Computer controlled by a PDP-11/34 system. The software provides direct control of the signal processor program and data memories via user-defined algorithms and allows for real-time interactive program execution. This approach is an alternative to musical data-entry systems. The musical and conceptual background differs however from some other language approaches to music.

A notable difference with much other synthesis software is the absence of the note concept. Neither the input nor the output is concerned with note lists or a score. Similarly a user's task is not the definition of an orchestra and its score but the definition of the changes which should take place at any or all of the available input registers. The difference is one of emphasis and of procedure.

Pile4 is a language in which musical algorithms can be expressed, compiled, and executed. This statement exposes the musical bias: musical composition is regarded as an algorithmic activity. The design of Pile4 is intended to facilitate those who want to work in this way. Although it fulfills certain definitions of a real-time system, no provision has been made for piano-style keyboards, unusual input devices, gestural interaction with performers, or other features which may be desirable in a performance system. I do not necessarily reject those kinds of systems. They just happen to fall outside the scope of my present work.

The convenience of the Pile4 programming language is enhanced by a number of integral utilities. They include a runtime library which allows simple real-time interaction with the user program during execution: any variable or list found either in the DMX-1000 or the PDP-11 may be redefined at will during program execution from the Ascii keyboard. A timer interrupt facility is provided. An extensive library of microcode for the DMX-1000 is available allowing the user to use several common synthesis techniques. By microcode I mean programs intended for the DMX-1000 program memory to control processor operation. Present microcode programs include realizations of waveshaping, the Karplus-Strong plucked string algorithm, various frequency modulation schemes including triple carriers and complex modulators, interpolating oscillators, table lookup oscillators, and linear digital oscillators. (A linear digital oscillator is a linear digital filter with the intended signal function as impulse response). Diverse schemes involving a-d processing as well as some very idiosyncratic synthesis modelling are also included. The microcode library is open-ended: new programs are easily added and can be used by a Pile4 program via parameter naming conventions. The expandibility of the library is of course beneficial to a user whose needs have not yet been met with current microcode. The integration in the Pile4 environment aids a microcode developer since much of the overhead necessary to test code is already present.

The appearance of the language is not radically different from other procedure-oriented languages. The importance is not its appearance but that by consistently implementing a number of simple insights it provides a degree of convenience in realizing certain categories of musical ideas in a particular hardware environment. A Pile4 program consists of accessing one or more microcode programs from the library and a user-written algorithm for controlling the microcode in time. The algorithm is expressed as a series of procedures taken either from the Pile4 procedural set or user defined (in terms of available Pile4 procedures, previous user definitions, or assembly language inserts). At intervals determined within the algorithm (commonly the satisfying of a cer-

tain condition within the hierarchy) one or more parameters of the microcode routine are updated. The updating is user-controlled at regular or irregular intervals. This simple mechanism permits the control of sound objects by coupling parameters with each other. In that case one always updates groups of parameters. But it also allows independent control of parameters thereby freeing one from the note concept and making many unusual things possible.

Pile4 procedures include some obviously necessary things to assign and control variables and lists, as well as procedures to facilitate data flow to the DMX-1000, control timer interrupts, specify variables and lists for real-time interaction, produce various kinds of random numbers, define lists according to piecewise line segments, define lists according to a user-specified algorithm, and to define lists according to tendency masks. Tendency is a selection principle for varying the boundaries in time within which a random choice can be made. It has been defined by Koenig in his composition programs PR2 and SSP and has been used by others including Barry Truax in some definitions of a compositional strategy.

A common working session would consist of defining a small kernel of an idea, compiling it, and fine-tuning the parameters during execution. The programmed idea is expanded, compiled, and fine-tuned. And so on. The resulting program is often a nested set of loops which update DMX-1000 parameters as needed. Both execution and turn-around time are fast enough to make this a feasible working method. The resulting source program contains all the necessary information to easily reproduce the results since sound is not produced by outputting a score but by executing procedures. This does impose a certain limit on the computational complexity. The tradeoff was made in favor of convenience and reproducibility which are important in an approach as empirical as this one.

Any parameter that can be defined as one or more locations in the DMX-1000 data memory can be algorithmically controlled. This implies that not only 'normal' parameters such as frequency can be organized according to an algorithmic structure but any other aspect of the sound can be controlled as well e.g. the over-modulation of the accumulated amplitude, a modulation of the table size or address in a lookup oscillator, the irregular distortion of the period length in an Karplus-Strong implementation, the substitution of incoming analog signals for the random table in Karplus-Strong, and so on and so forth. This is not just music as adventure since once a deviant,or if you prefer an 'inherently digital', process has been discovered, a range of procedures is available to control the parameter with any desired degree of complexity. Accidents can lead to interesting musical results. Should accidents not happen, it is of course sufficient for a user to rely on his musical insights to construct a program.

This discussion of the desirability of discovery in the compositional process is of course a bit of an aside. It is included to illustrate the scope and background of Pile4. Procedures can be used to fine-tune detail and to control larger structures as they unfold in time. Idiosyncratic applications are as convenient to realize as conventional strategies. But why be concerned with digitally-controllable idiosyncracies? Musical and historical reasons will be mentioned later in this paper. At this point I will just remind you of the facts of life: both in terms of money and of time, there is no reason not to use any of the digital synthesizers now being marketed if your primary interest is notes and the kinds of sounds which have now become traditional. The justification for bothering with computer software systems is that one wants to work with different strategies for producing sounds or structures. These are always, for at least a certain amount of time, idiosyncratic.

One of the major advantages of Pile4 is that it has already been implemented. It works. To illustrate this I would like to play a few sound examples.

The first example is a standard application of the Karplus-Strong Plucked String algorithm. (example 1).
The next example is also Karplus-Strong but with amplitude over-modulation. Each channel has three voices. Each voice approaches the maximum total amplitude value. Accumulating the three voices to send to the DAC results in this kind of distortion: (example 2).
The third example is a variation of Karplus-Strong. The basic idea of Karplus-Strong is, as you know, that a random table is filtered in such a way that the impression of a plucked string results. A new table must be loaded for every attack. In this example, a random table is not used. Instead a table is made by sampling an incoming instrumental signal. It is this oboe signal that is filtered. (example 3).

The next two examples demonstrate the importance of controlling modulations since they both use the same DMX-1000 microcode program. It is a standard FM implementation which also allows for distorting table lengths, waveforms,

filter settings, amplitude values, etc. (example 4) (example 5).

This approach to a procedural control language has several musical and historical roots. Previous work of mine (including the ASP programs and Pile2) involved real-time manipulation in the time domain. These programs explicitedly avoided referring to common musical quantities and concentrated on structures based on sample values and their distribution in time. In Pile4, I am not manipulating sample values and their distribution in time but the contents of one or more arbitrary registers in the signal processor and their distribution in time.

Various other composers (such as Brün, Koenig, and Xenakis) also were involved in what was once known as 'non-standard' sound synthesis. That term was a catch-all for many approaches which did not involve synthesis-by-rule. This work concerning the micro-structure of sound often used ideas previously applied to the macro-structure of a composition, e.g. Koenig's compositional programs based on selection procedures. Some non-computer-oriented composition also employs separate manipulation of relevant parameters, for example the work of many European serial composers.

Pile4 is an extension of past work of myself and others after confronting the opportunities presented by dedicated signal processing hardware. It shares with past work the interest in specifying a composition as an algorithm, in using random distribution principles, in controlling various parameters in accordance with a user-defined hierarchy, and in the desire to search for alternatives to melodic and harmonic conventions. An additional consideration: given current knowledge about the importance of modulation in sound production, it seems appropriate to provide control instructions which could facilitate the specification of desired modulations.

Those roots explain why I chose to make a procedural control language rather than implementing a musical data-entry system. Pile4 offers more generality and growth potential than data-entry systems. It provides more convenience than if one is given a signal processor, a general computer language, and is told to get to work. It is simple enough to be attractive to novice users and open enough to be a useful framework for demanding users. The approach could easily be transfered to modest hardware configurations involving for example the Texas Instruments TMS320 or various popular digital synthesizers. Pile4 is, of course, only one answer to the question of an appropriate environment for working with present digital signal processing technology.